ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

Лабораторная №2

По дисциплине «Архитектура вычислительных систем»

«Оценка производительности процессора»

Выполнил: студент гр. ИП-013

Копытина Т.А.

Проверил: ассистент кафедры ВС

Насонова А.О.

Новосибирск 2022г.

Оглавление

Цель лабораторной работы

1. Написать программу(ы) (benchmark) на языке C/C++/C# для оценки производительности процессора. В качестве набора типовых задач использовать либо минимум 3 функции выполняющих математические вычисления, либо одну функцию по работе с матрицами и векторами данных с несколькими типами данных. Можно использовать готовые функции из математической библиотеки (math.h) [3], библиотеки BLAS [4] (англ. Basic Linear Algebra г. Subprograms — базовые подпрограммы линейной алгебры) и/или библиотеки LAPACK [5] (Linear Algebra PACKage). Обеспечить возможность в качестве аргумента при вызове программы указать общее число испытаний для каждой типовой задачи (минимум 10). Входные данные для типовой задачи сгенерировать случайным образом.
2. С помощью системного таймера (библиотека time.h, функции clock() илиgettimeofday()) или с помощью процессорного регистра счетчика TSC реализовать оценку в секундах среднего времени испытания каждой типовой задачи. Оценить точность и погрешность (абсолютную и относительную) измерения времени (рассчитать дисперсию и среднеквадратическое отклонение).
3. Результаты испытаний в самой программе (или с помощью скрипта) сохранить в файл в формате CSV со следующей структурой: [PModel; Task; OpType; Opt; InsCount; Timer; Time; LNum; AvTime; AbsErr; RelErr; TaskPerf], где

PModel – Processor Model, модель процессора, на котором проводятся испытания;

Task – название выбранной типовой задачи (например, sin, log, saxpy, dgemv, sgemm и др.);

OpType – **Operand Type**, тип операндов, используемых при вычислениях типовой задачи;

Opt – Optimisations, используемы ключи оптимизации (None, O1, O2 и др.);

InsCount – Instruction Count, оценка числа инструкций при выполнении типовой задачи;

Timer – название функции обращения к таймеру (для измерения времени);

Time – время выполнения отдельного испытания;

LNum – Launch Numer, номер испытания типовой задачи.

AvTime –Average Time, среднее время выполнения типовой задачи из всех испытаний[секунды];

AbsError – Absolute Error, абсолютная погрешность измерения времени в секундах;

RelError – Relative Error, относительная погрешность измерения времени в %;

TaskPerf – Task Performance, производительность (быстродействие) процессора при выполнении типовой задачи.

Ход работы

Вначале создадим структуру, где запишем основные поля.

```
struct Stat()
```

Далее напишем функцию:

```
Stat SelectSort(),
```

которая будет типа `Stat`. В ней мы будем проверять оценку времени выполнения функции по характеристикам сортировки прямого выбора, которая имеет квадратичную трудоемкость.

Затем напишем функцию так же типа `Stat`:

```
Stat euler().
```

В которой опишем метод Эйлера и оценим время выполнения этой функции. Она имеет линейную трудоемкость.

Следующую функцию объявим, как

```
Stat fact(),
```

она так же будет типа `Stat`.

В которой будем оценивать время вычисления факториала. Она так же как и метод Эйлера имеет линейную трудоемкость.

Последней напишем функцию

```
string exec(const char *cmd),
```

в которой мы опишем некий метод decltype, который помогает выводить типы выражений и может сохранять их, не теряя const и &. Что поможет при записи в CSV файл.

Результаты работы

```
g++ -Wall -Werror main.cpp -o main
./main

DX = 0.000000
Sigma = 0.000207
Accuracy = 0.000006
infelicity = 0.000600%
Euler = 0.000218 sec.

DX = 0.001986
Sigma = 0.044561
Accuracy = 0.001188
infelicity = 0.118800%
Select sort 250 elements = 0.046905 sec.

DX = 0.000000
Sigma = 0.000314
Accuracy = 0.000016
infelicity = 0.001600%
Factorial 19, 10000 times = 0.000332 sec.
sh: 1: ./proc.sh: Permission denied
```

Рисунок 1 Вывод данных в консоль

	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт	Стандарт
1	PModel	Task	OpType	Opt	InsCount	Timer	Time	LNum	AvTime	AbsError	RelError	TaskPerf
2												
3	-	Euler	i	None	200000	clock_t / time.h	0.000215	10	0.0002182	6e-06	0.000600%	99.9994
4	-	Euler	i	None	200000	clock_t / time.h	0.000215	10	0.0002182	6e-06	0.000600%	99.9994
5	-	Euler	i	None	200000	clock_t / time.h	0.000221	10	0.0002182	6e-06	0.000600%	99.9994
6	-	Euler	i	None	200000	clock_t / time.h	0.000224	10	0.0002182	6e-06	0.000600%	99.9994
7	-	Euler	i	None	200000	clock_t / time.h	0.000226	10	0.0002182	6e-06	0.000600%	99.9994
8	-	Euler	i	None	200000	clock_t / time.h	0.000214	10	0.0002182	6e-06	0.000600%	99.9994
9	-	Euler	i	None	200000	clock_t / time.h	0.000215	10	0.0002182	6e-06	0.000600%	99.9994
10	-	Euler	i	None	200000	clock_t / time.h	0.000222	10	0.0002182	6e-06	0.000600%	99.9994
11	-	Euler	i	None	200000	clock_t / time.h	0.000216	10	0.0002182	6e-06	0.000600%	99.9994
12	-	Euler	i	None	200000	clock_t / time.h	0.000214	10	0.0002182	6e-06	0.000600%	99.9994
13	-	Select sort	i	None	2490	clock_t / time.h	0.046985	10	0.0469047	0.001188	0.118800%	99.8812
14	-	Select sort	i	None	2490	clock_t / time.h	0.048557	10	0.0469047	0.001188	0.118800%	99.8812
15	-	Select sort	i	None	2490	clock_t / time.h	0.047726	10	0.0469047	0.001188	0.118800%	99.8812
16	-	Select sort	i	None	2490	clock_t / time.h	0.046218	10	0.0469047	0.001188	0.118800%	99.8812
17	-	Select sort	i	None	2490	clock_t / time.h	0.047645	10	0.0469047	0.001188	0.118800%	99.8812
18	-	Select sort	i	None	2490	clock_t / time.h	0.046181	10	0.0469047	0.001188	0.118800%	99.8812
19	-	Select sort	i	None	2490	clock_t / time.h	0.046275	10	0.0469047	0.001188	0.118800%	99.8812
20	-	Select sort	i	None	2490	clock_t / time.h	0.046545	10	0.0469047	0.001188	0.118800%	99.8812
21	-	Select sort	i	None	2490	clock_t / time.h	0.046612	10	0.0469047	0.001188	0.118800%	99.8812
22	-	Select sort	i	None	2490	clock_t / time.h	0.046303	10	0.0469047	0.001188	0.118800%	99.8812
23	-	Factorial	x	None	1800000	clock_t / time.h	0.000341	10	0.000332	1.6e-05	0.001600%	99.9984
24	-	Factorial	x	None	1800000	clock_t / time.h	0.000323	10	0.000332	1.6e-05	0.001600%	99.9984
25	-	Factorial	x	None	1800000	clock_t / time.h	0.000329	10	0.000332	1.6e-05	0.001600%	99.9984
26	-	Factorial	x	None	1800000	clock_t / time.h	0.000324	10	0.000332	1.6e-05	0.001600%	99.9984
27	-	Factorial	x	None	1800000	clock_t / time.h	0.000323	10	0.000332	1.6e-05	0.001600%	99.9984

Рисунок 2 Результат работы в csv файле

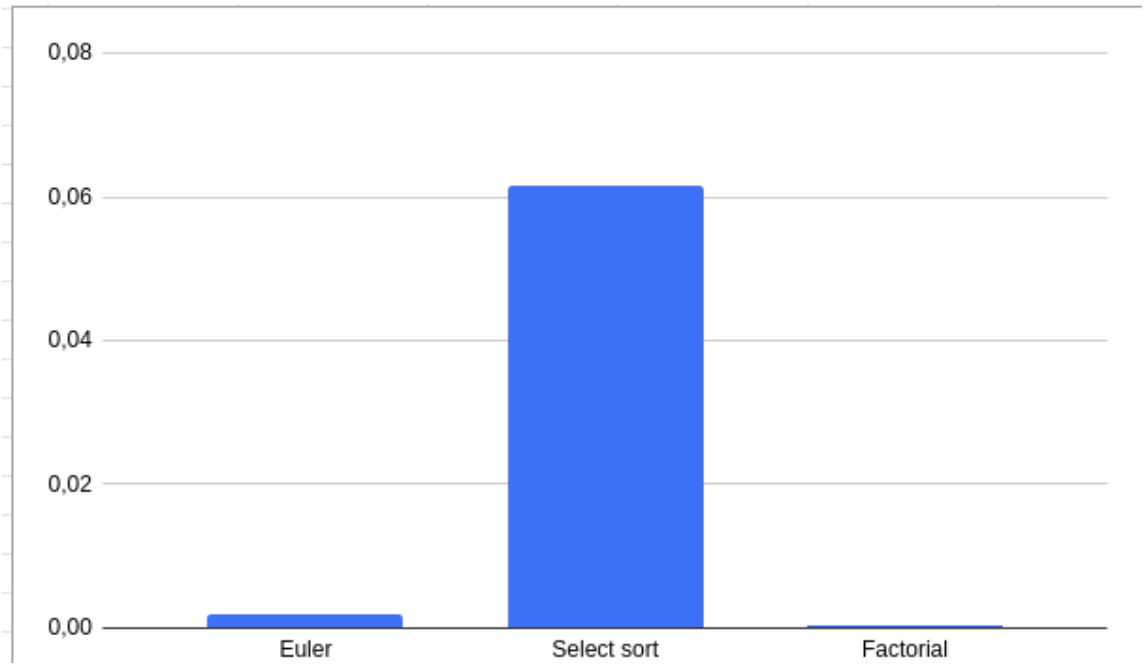

Рисунок 3 Построенная диаграмма среднего времени выполнения операции

На графике видно, что дольше всех работает SelectSort, который имеет квадратичную трудоемкость, в отличии от алгоритмов Эйлера и факториала, имеющих линейную трудоемкость в представленных реализациях.

Листинг

```
#include <iostream>

#include <ctime>

#include <cmath>

#include <chrono>

#include <fstream>

#include <string>

#include <cstdio>

#include <memory>

#include <stdexcept>

#include <sstream>

#include <array>

#include <cstdio>

#include <stdio.h>

#include <cstring>

using namespace std;


const int n = 250;

const int m = n;

const int k = n;

const int nums = 10;


struct Stat

{

    string Task;

    string OpType;

    string Opt;
```

```
    long long InsCount;
    string Timer;
    float Time[nums];
    int LNum;
    float AvTime;
    float AbsError;
    string RelError;
    float TaskPerf;
};

Stat SelectSort()
{
    int a[n];
    Stat stats;
    float EX = 0.0f;
    float alltime = 0.0f;
    float DX = 0.0f;
    float maxtime = 0.0f;
    float mintime = INFINITY;
    stats.InsCount = 0;
    for (int z = 0; z < nums; ++z)
    {
        for (int i = 0; i < n; ++i) a[i] = rand() % n;
        clock_t timer = clock();
        for (int i = 0; i < n - 1; i++) {
        int min_i = i;
        for(int asd = 0; asd < 1000; ++asd){
```

```
            for (int j = i + 1; j < n; j++) {
                if (a[j] < a[min_i]) {
                    min_i = j;
                }
            }
        }
        swap(a[i], a[min_i]);
        stats.InsCount++;
        }
        float time = (float)(clock() - timer) / CLOCKS_PER_SEC;
        stats.Time[z] = time;
        alltime += time;
        EX += (1 / (float)nums) * time;
        DX = pow(EX, 2) - ((1 / (float)nums) * pow(time, 2));
        if (time > maxtime)
            maxtime = time;
        if (time < mintime)
            mintime = time;
    }
    float accuracy = (maxtime - mintime) / 2;
    float Sigma = sqrt(DX);
    printf("\nDX = %f\nSigma = %f\nAccuracy = %f\ninfelicity =
%f%%\nSelect sort %d elements = %f sec.\n", DX, Sigma, accuracy,
(float)accuracy * 100, n, alltime / nums);
    stats.Task = "Select sort";
    stats.OpType = typeid(*a).name();
    stats.Opt = "None";
    stats.Timer = "clock_t / time.h";
```

```
    stats.LNum = nums;
    stats.AvTime = alltime / nums;
    stats.AbsError = accuracy;
    stats.RelError = (to_string((float)accuracy * 100) + '%');
    stats.TaskPerf = 100 - ((float)accuracy * 100);
    return stats;
}



Stat euler()
{
    Stat stats;
    float EX = 0.0f;
    float alltime = 0.0f;
    float DX = 0.0f;
    float maxtime = 0.0f;
    float mintime = INFINITY;
    stats.InsCount = 0;
    int result = n;
    int tmp = n;
    for (int z = 0; z < nums; z++)
    {
        clock_t timer = clock();
        for (int asd = 0; asd < 10000; ++asd){
            tmp = n;
         for (int i=2; i*i<=tmp; ++i)
               if (tmp % i == 0) {
```

```
                    while (tmp % i == 0)
                        tmp /= i;
                    result -= result / i;
                stats.InsCount++;
               }
         if (tmp > 1) result -= result / tmp;
        }
        float time = (float)(clock() - timer) / CLOCKS_PER_SEC;
        stats.Time[z] = time;
        alltime += time;
        EX += (1 / (float)nums) * time;
        DX = pow(EX, 2) - ((1 / (float)nums) * pow(time, 2));
        if (time > maxtime)
            maxtime = time;
        if (time < mintime)
            mintime = time;
    }
    float accuracy = (maxtime - mintime) / 2;
    float Sigma = sqrt(DX);
    printf("\nDX = %f\nSigma = %f\nAccuracy = %f\ninfelicity =
%f%%\nEuler = %f sec.\n", DX, Sigma, accuracy, (float)accuracy * 100,
alltime / nums);
    stats.Task = "Euler";
    stats.OpType = typeid(result).name();
    stats.Opt = "None";
    stats.Timer = "clock_t / time.h";
    stats.LNum = nums;
    stats.AvTime = alltime / nums;
```

```
    stats.AbsError = accuracy;
    stats.RelError = (to_string((float)accuracy * 100) + '%');
    stats.TaskPerf = 100 - ((float)accuracy * 100);

    return stats;
}

Stat fact()
{
    Stat stats;
    float EX = 0.0f;
    float alltime = 0.0f;
    float DX = 0.0f;
    float maxtime = 0.0f;
    float mintime = INFINITY;
    long long f = 1;
    stats.InsCount = 0;
    for (int z = 0; z < nums; z++)
    {
        f = 1;
        clock_t timer = clock();
        for (int x = 0; x < 10000; x++)
            for (long long i = 2; i <= 19; i++)
            {
                stats.InsCount++;
                f *= i;
            }
```

```
        float time = (float)(clock() - timer) / CLOCKS_PER_SEC;
        stats.Time[z] = time;
        alltime += time;
        EX += (1 / (float)nums) * time;
        DX = pow(EX, 2) - ((1 / (float)nums) * pow(time, 2));
        if (time > maxtime)
            maxtime = time;
        if (time < mintime)
            mintime = time;
    }
    float accuracy = (maxtime - mintime) / 2;
    float Sigma = sqrt(DX);
    printf("\nDX = %f\nSigma = %f\nAccuracy = %f\ninfelicity =
%f%%\nFactorial 19, 10000 times = %f sec.\n", DX, Sigma, accuracy,
(float)accuracy * 100, alltime / nums);
    stats.Task = "Factorial";
    stats.OpType = typeid(f).name();
    stats.Opt = "None";
    stats.Timer = "clock_t / time.h";
    stats.LNum = nums;
    stats.AvTime = alltime / nums;
    stats.AbsError = accuracy;
    stats.RelError = (to_string((float)accuracy * 100) + '%');
    stats.TaskPerf = 100 - ((float)accuracy * 100);

    return stats;
}
```

```
string exec(const char *cmd)
{
    array<char, 128> buffer;
    string result;
    unique_ptr<FILE, decltype(&pclose)> pipe(popen(cmd, "r"), pclose);
    while (fgets(buffer.data(), buffer.size(), pipe.get()) != nullptr)
    {
        result += buffer.data();
    }
    return result;
}

int main(int argc, char **argv)
{
    ofstream file;
    file.open("main.csv");
    Stat first = euler();
    Stat second = SelectSort();
    Stat third = fact();
    for (int i = 0; i < argc; i++)
    {
        if (!strcmp(argv[i], "-O1") || !strcmp(argv[i], "-O2") ||
!strcmp(argv[i], "-O3"))
        {
            first.Opt = argv[i];
            second.Opt = argv[i];
            third.Opt = argv[i];
            break;
```

```
        }
    }
    stringstream os;
    stringstream os2;
    stringstream os3;
    file << "PModel, Task, OpType, Opt, InsCount, Timer, Time, LNum,
AvTime, AbsError, RelError, TaskPerf\n" << exec("./proc.sh") << "\n";
    for (float i : first.Time) {
        os << i << " ";
        string sTime(os.str());
        file << "-, " << first.Task << ", " << first.OpType << ", " <<
first.Opt << ", " << first.InsCount << ", " << first.Timer << ", " <<
sTime << ", " << first.LNum << ", " << first.AvTime << ", " <<
first.AbsError << ", " << first.RelError << ", " << first.TaskPerf <<
"\n";
        os.str("");
        sTime.clear();
    }
    for (float i : second.Time) {
        os << i << " ";
        string sTime(os.str());
        file << "-, " << second.Task << ", " << second.OpType << ", "
<< second.Opt << ", " << second.InsCount << ", " << second.Timer << ",
" << sTime << ", " << second.LNum << ", " << second.AvTime << ", " <<
second.AbsError << ", " << second.RelError << ", " << second.TaskPerf
<< "\n";
        os.str("");
        sTime.clear();
    }
    for (float i : third.Time) {
```

```
        os << i << " ";
        string sTime(os.str());
        file << "-, " << third.Task << ", " << third.OpType << ", " <<
third.Opt << ", " << third.InsCount << ", " << third.Timer << ", " <<
sTime << ", " << third.LNum << ", " << third.AvTime << ", " <<
third.AbsError << ", " << third.RelError << ", " << third.TaskPerf <<
"\n";
        os.str("");
        sTime.clear();
    }
    file.close();
    return 0;
}
```